

# ALGORITMOS PARA ESTATÍSTICAS DE ORDEM

MO417 - Complexidade de  
Algoritmos I

Santiago Valdés Ravelo  
[https://ic.unicamp.br/~santiago/  
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)


04/24

12



UNICAMP





*“A maioria das pessoas usa estatísticas como um bêbado usa um poste de luz; mais para apoio do que para iluminação.”*

Andrew Lang.



# ESTADÍSTICAS DE ORDEM



## Estatísticas de ordem

Estamos interessados no seguinte problema

### Problema (Problema da seleção)

**Entrada:** Um vetor  $A$  com  $n$  números reais e um inteiro  $i$ .

**Saída:** O  $i$ -ésimo menor elemento de  $A$ .

Casos particulares importantes:

- ▶ Mínimo:  $i = 1$ .
- ▶ Máximo:  $i = n$ .
- ▶ Mediana:  $i = \lfloor \frac{n+1}{2} \rfloor$  (mediana inferior).
- ▶ Mediana:  $i = \lceil \frac{n+1}{2} \rceil$  (mediana superior).



MÍNIMO E MÁXIMO



## Mínimo

---

**Algoritmo:** MÍNIMO( $A, n$ )

---

```
1 mínimo  $\leftarrow A[1]$ 
2 para  $j \leftarrow 2$  até  $n$ 
3   se mínimo  $> A[j]$ 
4     mínimo  $\leftarrow A[j]$ 
5 devolva mínimo
```

---

Análise:

- ▶ Realiza  $n - 1$  comparações.
- ▶ É possível fazer menos? Não. Como provar?



## Cota inferior para problema do mínimo

### Problema

**Entrada:** Um vetor  $A[1 \dots n]$ .

**Saída:** O menor elemento de  $A$ .

O algoritmo trivial executa  $n - 1$  comparações:

- ▶ É o melhor algoritmo **baseado em comparações**.
- ▶ Para ver isso, vamos investigar um algoritmo **genérico**.



## Algoritmo genérico para o problema do mínimo

Para um algoritmo, defina uma coleção  $\mathcal{A}$  de pares  $(i, j)$

- ▶ Um par  $(i, j)$  representa uma  $A[i] \geq A[j]$ .
- ▶ Dizemos que  $i$  perdeu a disputa.
- ▶ Adicionamos um par para cada comparação executada.

---

**Algoritmo:** MÍNIMO( $A, n$ )

---

- 1  $\mathcal{A} \leftarrow \emptyset$
- 2 **enquanto** o algoritmo não encontrou o mínimo
- 3     escolha índices  $i$  e  $j$  em  $\{1, \dots, n\}$ , que Alg compara
- 4     **se**  $A[i] \geq A[j]$
- 5          $\mathcal{A} \leftarrow \mathcal{A} \cup \{(i, j)\}$
- 6     **senão**
- 7          $\mathcal{A} \leftarrow \mathcal{A} \cup \{(j, i)\}$
- 8 **devolva** o mínimo encontrado





## Cota inferior para problema do mínimo

Considere a coleção de pares  $\mathcal{A}$ :

- ▶ O mínimo nunca perde uma disputa.
- ▶ Cada um dos outros deve perder pelo menos uma vez.
- ▶ Assim  $\mathcal{A}$  tem pelo menos  $n - 1$  pares.

Concluimos:

- ▶ Todo algoritmo para o problema do mínimo baseado em comparações executa **pelo menos**  $n - 1$  comparações.
- ▶ O algoritmo trivial dado é ótimo.



## Mínimo e máximo

---

**Algoritmo:** MIN-MAX( $A, n$ )

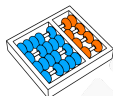
---

```
1 mínimo ← máximo ←  $A[1]$ 
2 para  $j \leftarrow 2$  até  $n$ 
3   se  $A[j] < \text{mínimo}$ 
4     | mínimo ←  $A[j]$ 
5   se  $A[j] > \text{máximo}$ 
6     | máximo ←  $A[j]$ 
7 devolva(mínimo, máximo)
```

---

Análise:

- ▶ Realiza  $2(n - 1)$  comparações.
- ▶ É possível fazer melhor!



## Mínimo e máximo (melhorado)

Ideia para melhorar algoritmo:

- ▶ Inicializamos mínimo e máximo:
  - ▶ Com o primeiro elemento se  $n$  for ímpar.
  - ▶ Com o mínimo e o máximo dos dois primeiros se  $n$  for par.
- ▶ Comparamos os demais elementos em **pares**:
  - ▶ O menor deles com mínimo.
  - ▶ O maior deles com máximo.
- ▶ Fazemos 3 comparações para cada 2 elementos.

Análise:

- ▶ O número de comparações é dado por:

$$\begin{cases} 3\lfloor n/2 \rfloor & \text{se } n \text{ for ímpar} \\ 3(n-2)/2 + 1 & \text{se } n \text{ for par} \end{cases}$$

- ▶ Não é possível fazer melhor (exercício).



# PROBLEMA DA SELEÇÃO



## Algoritmo para problema da seleção

Podemos selecionar o  $i$ -ésimo elemento ordenando o vetor.

---

**Algoritmo:** SELECT-ORD( $A, n, i$ )

---

- 1 ORDENE( $A, n$ )
  - 2 **devolva**  $A[i]$
- 

Análise:

- ▶ Usamos MERGE-SORT ou HEAP-SORT como ORDENE.
- ▶ O algoritmo realiza  $O(n \log n)$  comparações.

É possível fazer melhor que isso?

- ▶ Achamos mínimo e máximo com  $O(n)$  comparações.
- ▶ Vamos tentar descobrir o  $i$ -ésimo em **tempo linear**.



## Relembrando particionamento

### Problema

**Entrada:** Um vetor  $A[p \dots r]$ .

**Saída:** Um índice  $q$ , com  $p \leq q \leq r$ , tal que

$$A[p \dots q - 1] \leq A[q] < A[q + 1 \dots r].$$

Entrada:

	$p$									$r$
A	99	33	55	77	11	22	88	66	33	44

Saída:

	$p$			$q$						$r$
A	33	11	22	33	44	55	99	66	77	88



## Relembrando PARTICIONE

---

**Algoritmo:** PARTICIONE( $A, p, r$ )

---

```
1  $x \leftarrow A[r]$   ▷  $x$  é o pivô
2  $i \leftarrow p - 1$ 
3 para  $j \leftarrow p$  até  $r - 1$ 
4   se  $A[j] \leq x$ 
5      $i \leftarrow i + 1$ 
6      $A[i] \leftrightarrow A[j]$ 
7  $A[i+1] \leftrightarrow A[r]$ 
8 devolva  $i + 1$ 
```

---



## Selecionando via particionamento

Ideia:

- ▶ Particionamos o vetor de forma que:

$$A[1 \dots q - 1] \leq A[q] < A[q + 1 \dots n]$$

- ▶ Verificamos o índice  $q$  do pivô:

1. Se  $i = q$ , então o  $i$ -ésimo menor é  **$A[q]$** !
2. Se  $i < q$ , então o  $i$ -ésimo menor está em  **$A[1 \dots q - 1]$** .
3. Se  $i > q$ , então o  $i$ -ésimo menor está em  **$A[q + 1 \dots n]$** .





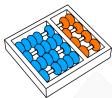
## Algoritmo recursivo

Subproblema recursivo:

- ▶ A entrada é um subvetor  $A[p \dots r]$  e posição  $i = 1, 2, \dots$
- ▶ A saída é o  $i$ -ésimo menor valor **nesse subvetor**.

Exemplo:

- ▶ Considere um subvetor  $A[9 \dots 18]$  e posição  $i = 5$ .
- ▶ Se  $A$  estivesse ordenado, a saída deveria ser  $A[13]$ .
- ▶ Suponha que após particionar, o pivô esteja em  $q = 11$ .
- ▶ A posição do pivô é  $k = q - p + 1 = 11 - 9 + 1 = 3$ .
- ▶ Então  $A[q]$  é terceiro menor valor nesse subvetor.
- ▶ Assim, procuramos o segundo menor depois do pivô.



## Algoritmo baseado em particionamento

---

### Algoritmo: SELECT-NL( $A, p, r, i$ )

---

```
1 se  $p = r$ 
2   | devolva  $A[p]$ 
3  $q \leftarrow$  PARTICIONE( $A, p, r$ )
4  $k \leftarrow q - p + 1$ 
5 se  $i = k$ 
6   | devolva  $A[q]$ 
7 senão
8   | se  $i < k$ 
9     | devolva SELECT-NL( $A, p, q - 1, i$ )
10  | senão
11  | devolva SELECT-NL( $A, q + 1, r, i - k$ )
```

- 
- ▶  $p$  e  $r$  são os índices de limite do vetor.
  - ▶  $k$  é a posição do pivô no vetor considerado.



## Análise do algoritmo melhorado

SELECT-NL( $A, p, r, i$ )	Tempo
1 <b>se</b> $p = r$	$\Theta(1)$
2 <b>devolva</b> $A[p]$	$O(1)$
3 $q \leftarrow \text{PARTICIONE}(A, p, r)$	$\Theta(n)$
4 $k \leftarrow q - p + 1$	$\Theta(1)$
5 <b>se</b> $i = k$	$\Theta(1)$
6 <b>devolva</b> $A[q]$	$O(1)$
7 <b>senão</b>	$O(1)$
8 <b>se</b> $i < k$	$O(1)$
9 <b>devolva</b> SELECT-NL( $A, p, q - 1, i$ )	$T(k - 1)$
10 <b>senão</b>	$O(1)$
11 <b>devolva</b> SELECT-NL( $A, q + 1, r, i - k$ )	$T(n - k)$

Seja  $n = r - p + 1$ :

- ▶ No pior caso  $T(n) = \max\{T(k - 1), T(n - k)\} + \Theta(n)$ .
- ▶ Já sabemos que  $T(n) = \Theta(n^2)$ .



## Análise do algoritmo melhorado (cont)

Comparando:

- ▶ Sabemos que SELECT-ORD gasta tempo  $O(n \log n)$ .
- ▶ Mas, no pior caso, SELECT-NL tem complexidade  $\Theta(n^2)$ .

Seria melhor usar SELECT-ORD?

- ▶ Não, SELECT-NL é eficiente na prática.
- ▶ No **caso médio**, ele tem complexidade  $O(n)$ .



## Versão aleatorizada

- ▶ O pior caso ocorre devido a escolhas infelizes do pivô.
- ▶ Podemos minimizar isso usando aleatoriedade.
- ▶ Vamos reutilizar PARTICIONE-ALEATORIZADO.

---

**Algoritmo:** PARTICIONE-ALEATORIZADO( $A, p, r$ )

---

- 1  $i \leftarrow \text{RANDOM}(p, r)$
  - 2  $A[i] \leftrightarrow A[r]$
  - 3 **devolva** PARTICIONE( $A, p, r$ )
-



## Algoritmo aleatorizado

---

**Algoritmo:** SELECT-ALEATORIZADO( $A, p, r, i$ )

---

```
1 se  $p = r$ 
2   ┌ devolva  $A[p]$ 
3  $q \leftarrow$  PARTICIONE-ALEATORIZADO( $A, p, r$ )
4  $k \leftarrow q - p + 1$ 
5 se  $i = k$ 
6   ┌ devolva  $A[q]$ 
7 senão
8   ┌ se  $i < k$ 
9     ┌ devolva SELECT-ALEATORIZADO( $A, p, q - 1, i$ )
10  senão
11  ┌ devolva SELECT-ALEATORIZADO( $A, q + 1, r, i - k$ )
```

---



## Análise do tempo esperado

PARTICIONE-ALEATORIZADO devolve um pivô  $q$  aleatoriamente:

- ▶ O tamanho do subvetor  $A[p \dots q]$  pode ser  $1, 2, \dots, n$ .
- ▶ Para cada possibilidade  $k$ , defina uma variável indicadora:

$$X_k = \begin{cases} 1 & \text{se } A[p \dots q] \text{ tem } \mathbf{exatamente} \ k \text{ elementos} \\ 0 & \text{caso contrário} \end{cases}$$



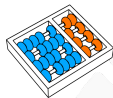
## Análise do tempo esperado

- ▶ Seja  $n = r - p + 1$  o tamanho do vetor de entrada.
- ▶ Podemos limitar o tempo de execução como:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 0, 1 \\ \sum_{k=1}^n X_k T(\max\{k-1, n-k\}) + \Theta(n) & \text{se } n \geq 2 \end{cases}$$

- ▶ Note que  $T(n)$  é uma variável aleatória.
- ▶ Queremos calcular  $E[T(n)]$ .





## Análise do tempo esperado

$$\begin{aligned}
 E[T(n)] &\leq E \left[ \sum_{k=1}^n X_k T(\max\{k-1, n-k\}) + an \right] \\
 &\leq \sum_{k=1}^n E[X_k] E[T(\max\{k-1, n-k\})] + an \\
 &\leq \sum_{k=1}^n \frac{1}{n} E[T(\max\{k-1, n-k\})] + an \\
 &\leq \frac{2}{n} \sum_{j=\lceil n/2 \rceil}^{n-1} E[T(j)] + an.
 \end{aligned}$$

Pois:

$$\max\{k-1, n-k\} = \begin{cases} k-1 & \text{se } k > \lceil n/2 \rceil, \\ n-k & \text{se } k \leq \lceil n/2 \rceil. \end{cases}$$



## Análise do tempo esperado

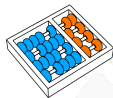
Vamos demonstrar por indução que  $E[T(n)] \leq cn$

$$E[T(n)] \leq \frac{2}{n} \sum_{j=\lfloor n/2 \rfloor}^{n-1} E[T(j)] + an$$

$$\stackrel{\text{h.i.}}{\leq} \frac{2}{n} \sum_{j=\lfloor n/2 \rfloor}^{n-1} cj + an$$

$$= \frac{2c}{n} \left( \sum_{j=1}^{n-1} j - \sum_{j=1}^{\lfloor n/2 \rfloor - 1} j \right) + an$$

$$= \frac{2c}{n} \left( \frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2} \right) + an.$$



## Análise do tempo esperado

$$\begin{aligned}
 E[T(n)] &\leq \frac{2c}{n} \left( \frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2} \right) + an \\
 &\leq \frac{2c}{n} \left( \frac{(n-1)n}{2} - \frac{(n/2 - 2)(n/2 - 1)}{2} \right) + an \\
 &= \frac{c}{n} \left( \frac{3n^2}{4} + \frac{n}{2} - 2 \right) + an \\
 &\leq \frac{3cn}{4} + \frac{c}{2} + an \\
 &= cn - \left( \frac{cn}{4} - \frac{c}{2} - an \right) \leq cn.
 \end{aligned}$$

Se  $c > 4a$  e  $n \geq 2c/(c - 4a)$ .



ALGORITMO LINEAR



## Algoritmo linear para seleção

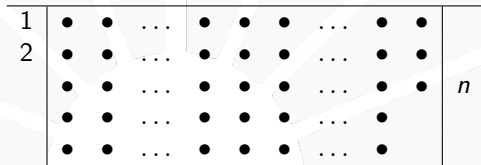
Veremos um algoritmo linear para o problema da seleção.

- ▶ Chamaremos de **algoritmo BFPRT**.
- ▶ Os autores Blum, Floyd, Pratt, Rivest e Tarjan.
- ▶ Vamos supor que os elementos em  $A$  são distintos.

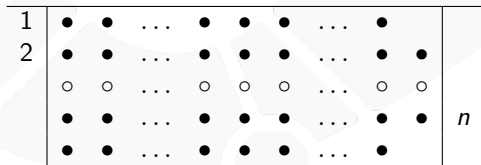


## Algoritmo BFPRT

1. Divida os  $n$  elementos em  $\lfloor \frac{n}{5} \rfloor$  subconjuntos de 5 elementos e um subconjunto de  $n \bmod 5$  elementos.



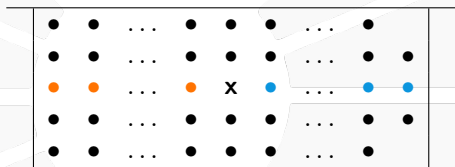
2. Encontre a mediana de cada um dos  $\lfloor \frac{n}{5} \rfloor$  subconjuntos.



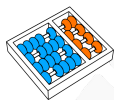


## Algoritmo BFPRT

3. Determine, recursivamente, a **mediana das medianas**  $x$  dos subconjuntos de no máximo 5 elementos.



- ▶ **Note que o algoritmo não ordena as medianas!**



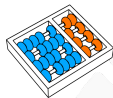
## Algoritmo BFPRT

4. Usando  $x$  como pivô, particione o conjunto original  $A$  criando dois subconjuntos  $A_{<}$  e  $A_{>}$ , em que:
- ▶  $A_{<}$  contém os elementos menores que  $x$ .
  - ▶  $A_{>}$  contém os elementos maiores que  $x$ .

Se a posição final de  $x$  após o particionamento for  $k$ , então

$$|A_{<}| = k - 1 \quad \text{e} \quad |A_{>}| = n - k.$$





## Algoritmo BFPRT

5. Finalmente, para encontrar o  $i$ -ésimo menor elemento do conjunto, compare  $i$  com a posição  $k$  de  $x$  após o particionamento:
- ▶ Se  $i = k$ , o elemento procurado é  $x$ .
  - ▶ Se  $i < k$ , procure recursivamente o  $i$ -ésimo de  $A_{<}$ .
  - ▶ Se  $i > k$ , procure recursivamente o  $(i - k)$ -ésimo de  $A_{>}$ .



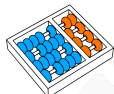
## Análise do algoritmo BFPRT

Seja  $T(n)$  a complexidade de tempo no pior caso:

- |                                                  |                        |
|--------------------------------------------------|------------------------|
| 1. Divisão em subconjuntos de 5 elementos.       | $\Theta(n)$            |
| 2. Encontrar a mediana de cada subconjunto.      | $\Theta(n)$            |
| 3. Encontrar $x$ , a mediana das medianas.       | $T(\lceil n/5 \rceil)$ |
| 4. Particionamento com pivô $x$ .                | $O(n)$                 |
| 5. Encontrar o $i$ -ésimo menor de $A_{<}$ ,     | $T(k - 1)$             |
| ou encontrar o $i - k$ -ésimo menor de $A_{>}$ . | $T(n - k)$             |

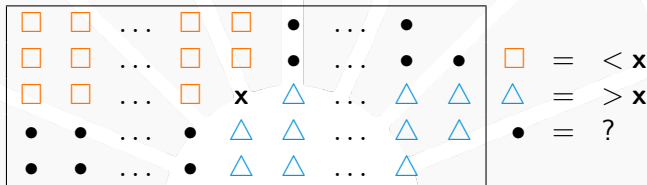
Obtemos a recorrência para o pior caso:

$$T(n) = T(\lceil n/5 \rceil) + T(\max\{k - 1, n - k\}) + \Theta(n).$$



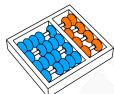
## Análise do algoritmo BFPRT

O diagrama abaixo classifica os elementos da última figura.



Podemos contar o número de elementos maiores que x:

- ▶ Contém pelo menos os triângulos  $\triangle$  da figura.
- ▶ Há no mínimo  $\frac{3n}{10} - 6$  tantos elementos:
  - ▶ Há  $\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil$  grupos com 3 elementos maiores que x.
  - ▶ Exceto talvez o último e o que contém x.
  - ▶ Assim o número de elementos é:  $3 \left( \lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil - 2 \right) \geq \frac{3n}{10} - 6$ .



## Análise do algoritmo BFPRT

Repetindo o argumento para os quadrados  $\square$ :

- ▶ Há pelo menos  $\frac{3n}{10} - 6$  elementos menores que  $x$ .
- ▶ Portanto,

$$\max\{k - 1, n - k\} \leq n - \left(\frac{3n}{10} - 6\right) \leq \frac{7n}{10} + 6.$$

Encontramos uma recorrência limitada por:

$$T(n) \leq \begin{cases} \Theta(1) & n \leq 140, \\ T(\lceil n/5 \rceil) + T(\lfloor 7n/10 \rfloor + 6) + \Theta(n) & n > 140. \end{cases}$$

- ▶ O limiar 140 foi escolhido para as contas funcionarem.
- ▶ A solução da recorrência é  $T(n) \in \Theta(n)$ .



## Análise do algoritmo BFPRT

Vamos demonstrar por indução que  $T(n) \leq cn$ :

$$T(n) \leq T(\lceil n/5 \rceil) + T(\lfloor 7n/10 \rfloor + 6) + an$$

**h.i.**

$$\leq c\lceil n/5 \rceil + c(\lfloor 7n/10 \rfloor + 6) + an$$

$$\leq c(n/5 + 1) + c(7n/10 + 6) + an$$

$$= 9cn/10 + 7c + an$$

$$= cn + (-cn/10 + 7c + an).$$

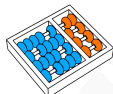


## Análise do algoritmo BFPRT

- ▶ Queremos obter a desigualdade:

$$\begin{aligned}T(n) &= cn + (-cn/10 + 7c + an) \\ &\leq cn.\end{aligned}$$

- ▶ Para isso, queremos  $(-cn/10 + 7c + an) \leq 0$ .
- ▶ Isolando  $c$ , vemos que é equivalente a  $c \geq 10an/(n - 70)$ .
- ▶ Note que  $n/(n - 70) \leq 2$  justamente para  $n \geq 140$ .
- ▶ Portanto, para  $n \geq 140$  basta escolher  $c \geq 20a$ .



## Pseudocódigo do algoritmo BFPRT

---

**Algoritmo:** SELECT-BFPRT( $A, p, r, i$ )

---

```

1 se  $p = r$ 
2   | devolva  $p$ 
3  $q \leftarrow$  PARTICIONE-BFPRT( $A, p, r$ )
4  $k \leftarrow q - p + 1$ 
5 se  $i = k$ 
6   | devolva  $q$ 
7 senão
8   | se  $i < k$ 
9     | devolva SELECT-BFPRT( $A, p, q - 1, i$ )
10  | senão
11  | devolva SELECT-BFPRT( $A, q + 1, r, i - k$ )

```

---

- ▶ Essa versão devolve o índice do elemento, **NÃO** o valor.
- ▶ Usaremos o índice em PARTICIONE-BFPRT.



## Pseudocódigo do particionamento de BFPRT

---

### Algoritmo: PARTICIONE-BFPRT( $A, p, r$ )

---

```

1  $\ell \leftarrow p, j \leftarrow p$ 
2 enquanto  $j + 4 < r$ 
3   ORDENE( $A, j, j + 4$ )
4    $A[\ell] \leftrightarrow A[j + 2]$ 
5    $j \leftarrow j + 5, \ell \leftarrow \ell + 1$ 
6 ORDENE( $A, j, r$ )
7  $A[\ell] \leftrightarrow A[(j + r)/2]$ 
8  $m \leftarrow \ell - p + 1$ 
9  $k \leftarrow$  SELECT-BFPRT( $A, p, \ell, \lfloor m/2 \rfloor$ )
10  $A[k] \leftrightarrow A[r]$ 
11 devolva PARTICIONE( $A, p, r$ )

```

---

- ▶ O índice  $j$  marca o início de cada grupo de 5 elementos.
- ▶ O índice  $\ell$  indexa as medianas dos grupos.
- ▶ As medianas são colocadas no começo do vetor (linhas 4 e 7).
- ▶ Na linha 9, descobrimos o índice da mediana das medianas.
- ▶ Na linha 11 particionamos com essa mediana.



# ALGORITMOS PARA ESTATÍSTICAS DE ORDEM

MO417 - Complexidade de  
Algoritmos I

Santiago Valdés Ravelo  
[https://ic.unicamp.br/~santiago/  
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

04/24

12



UNICAMP

